```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <string.h>
#include <stdlib.h>

//A function to initialize the microcontroller
void initializeMCU(void);

//Buffer to store the measured audio levels
uint8_t buffer[2048];
//Pointer to the current spot in the buffer to put the newly measured audio data
uint16_t new_data = 0;
//Pointer to the least-delayed output value
uint16_t out;
//Variable time delay - changing this value changes the delay between each
//speaker. It is measured in units of 1/44.15kHz = 22.7us
volatile int16_t td=0;

//Main function
int main(void)
{
        //Temporary variable to store output from internal ADC
        uint8_t temp;
        //Initilize the microcontroller
        initializeMCU();
        //Main while loop
        while(1){
                //Initilize an ADC reading (in case it hasn't been initialized
                //from the last run)
                ADCSRA |= (1<<ADSC);
                //Read the current ADC value into a temporary variable
                temp = ADCH;
                //Set the delay based on the ADC value, it is set between -50
                //and 50
                if(temp<28) td = -50;
                else if (temp < 56) td = -37;
                else if (temp < 85) td = -25;
                else if (temp < 113) td = -12;
                else if (temp < 142) td = 0;
                else if (temp < 170) td = 12;
                else if (temp < 198) td = 25;
                else if (temp < 226) td = 37;
                else td = 50;
        }
}


//MCU setup
void initializeMCU(void){

        // timer 1 ticks at 44.15 kHz
        OCR1A = 453 ; //453 ticks
        TIMSK1 = (1<<OCIE1A);
        TCCR1B = 0x09;  //full speed; clear-on-match
        TCCR1A = 0x00;  //turn off pwm and oc lines

        //Set up SPI
        //SPIE = 0, no interrupt
        //SPE = 1, enable SPI
        //DORD = 0, MSB first
        //MSTR = 1, master SPI mode
        //CPOL = 0, sck is low when idle
        //CPHA = 0, sample on leading edge, setup on falling edge
        //SPR1 = 0, SPR0 = 1, two bits for fosc/16
        SPCR = (1<<SPE)|
               (1<<MSTR)|
               (0<<CPOL)|
               (1<<CPHA)|
               (1<<SPR0);
        //Set the SCK, MOSI, and CS DDR to output
        DDRB = (1<<PB4)|(1<<PB5)|(1<<PB7);
        //Set Port C to outputs for controlling the DAC output
        DDRC = 0xFF;
        //Set the ADC input on port A to an input, the rest to outputs
        DDRA = ~(1<<PA2);
        //Set port D to outputs for debugging
        DDRD = 0xFF;
        //Set up the ADC to run on PA2 and be left adjusted
        ADMUX = (1<<REFS0)|(1<<ADLAR)|(1<<MUX1);
        //Free running mode
        ADCSRB = 0x00;
        //Enable and start the first conversion
        ADCSRA = (1<<ADEN)|(1<<ADSC)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
        // turn on all ISRs
        sei() ;
}

//Sound sampling and generation interrupt
ISR (TIMER1_COMPA_vect) // Fs=44150
{
        //Set PORTD 1 to output to measure CPU loading
        PORTD |= (1<<PD0);
        //Take the absolute value of the delay for later use
        uint16_t td_t=abs(td);
        //Temporary storage variable for SPI communications
        uint8_t temp = 0;
        //Pull chip select low
        PORTB &= ~(1<<PB4);
        //Send an SPI message. The ADC has no input, so we are sending this
        //just to force the serial clock to run
        SPDR = 0x00;
        //Start to set DAC outputs, and do so basaed on whether or not td
        //is greater than 0. The sign of td defines which side of the array
        //is delayed, and which is most current
        if(td > 0)
        {
                //In this code, set speakers 1-8 to their appropriate output
                //values. The bitwise and with 0x07FF is a modulo addressing
                //system to avoid bounds checking
                PORTC = buffer[out&0x07FF];
                PORTA = (1<<PA4);
                PORTA = (1<<PA3)|(1<<PA4);
                PORTC = buffer[(out+td_t)&0x07FF];
                PORTA = (1<<PA7)|(1<<PA4);
                PORTA = (1<<PA3)|(1<<PA4);
                PORTC = buffer[(out+2*td_t)&0x07FF];
                PORTA = (1<<PA6)|(1<<PA4);
                PORTA = (1<<PA3)|(1<<PA4);
                PORTC = buffer[(out+3*td_t)&0x07FF];
                PORTA = (1<<PA6)|(1<<PA7)|(1<<PA4);
                PORTA = (1<<PA3)|(1<<PA4);
```

```
		PORTC = buffer[(out+4*td_t)&0x07FF];
		PORTA = (1<<PA5)|(1<<PA4);
		PORTA = (1<<PA3)|(1<<PA4);
		PORTC = buffer[(out+5*td_t)&0x07FF];
		PORTA = (1<<PA5)|(1<<PA7)|(1<<PA4);
		PORTA = (1<<PA3)|(1<<PA4);
		PORTC = buffer[(out+6*td_t)&0x07FF];
		PORTA = (1<<PA5)|(1<<PA6)|(1<<PA4);
		PORTA = (1<<PA3)|(1<<PA4);
		PORTC = buffer[(out+7*td_t)&0x07FF];
		PORTA = (1<<PA5)|(1<<PA6)|(1<<PA7)|(1<<PA4);
		PORTA = (1<<PA3)|(1<<PA4);
	}
	else
	{
		//Set ports 12-5 to their correct outputs. This is the case
		//when td is negative.
		PORTC = buffer[out&0x07FF];
		PORTA = (1<<PA6)|(1<<PA7)|(1<<PA3);
		PORTA = (1<<PA4)|(1<<PA3);
		PORTC = buffer[(out+td_t)&0x07FF];
		PORTA = (1<<PA6)|(1<<PA3);
		PORTA = (1<<PA3)|(1<<PA4);
		PORTC = buffer[(out+2*td_t)&0x07FF];
		PORTA = (1<<PA7)|(1<<PA3);
		PORTA = (1<<PA3)|(1<<PA4);
		PORTC = buffer[(out+3*td_t)&0x07FF];
		PORTA = (1<<PA3);
		PORTA = (1<<PA3)|(1<<PA4);
		PORTC = buffer[(out+4*td_t)&0x07FF];
		PORTA = (1<<PA5)|(1<<PA6)|(1<<PA7)|(1<<PA4);
		PORTA = (1<<PA3)|(1<<PA4);
		PORTC = buffer[(out+5*td_t)&0x07FF];
		PORTA = (1<<PA5)|(1<<PA6)|(1<<PA4);
		PORTA = (1<<PA3)|(1<<PA4);
		PORTC = buffer[(out+6*td_t)&0x07FF];
		PORTA = (1<<PA5)|(1<<PA7)|(1<<PA4);
		PORTA = (1<<PA3)|(1<<PA4);
		PORTC = buffer[(out+7*td_t)&0x07FF];
		PORTA = (1<<PA5)|(1<<PA4);
		PORTA = (1<<PA3)|(1<<PA4);
	}
	//Wait for the first SPI communication to complete
	while (!(SPSR & (1<<SPIF)));
	//Store the output from the ADC in a temporary variable. This will
	//be two zeros, followed by the 6MSB of the ADC
	temp = SPDR;
	//Start another SPI communication by writing to SPDR
	SPDR = 0x00;
	//Again, set the outputs for the last four DACs depending on the sign
	//of the delay
	if(td > 0)
	{
		//Set outputs 8-12
		PORTC = buffer[(out+8*td_t)&0x07FF];
		PORTA = (1<<PA3);
		PORTA = (1<<PA3)|(1<<PA4);
		PORTC = buffer[(out+9*td_t)&0x07FF];
		PORTA = (1<<PA7)|(1<<PA3);
		PORTA = (1<<PA3)|(1<<PA4);
		PORTC = buffer[(out+10*td_t)&0x07FF];
```

```
                PORTA = (1<<PA6)|(1<<PA3);
                PORTA = (1<<PA3)|(1<<PA4);
                PORTC = buffer[(out+11*td_t)&0x07FF];
                PORTA = (1<<PA6)|(1<<PA7)|(1<<PA3);
                PORTA = (1<<PA4)|(1<<PA3);
        }
        else
        {
                //Set outputs 4-1
                PORTC = buffer[(out+8*td_t)&0x07FF];
                PORTA = (1<<PA6)|(1<<PA7)|(1<<PA4);
                PORTA = (1<<PA3)|(1<<PA4);
                PORTC = buffer[(out+9*td_t)&0x07FF];
                PORTA = (1<<PA6)|(1<<PA4);
                PORTA = (1<<PA3)|(1<<PA4);
                PORTC = buffer[(out+10*td_t)&0x07FF];
                PORTA = (1<<PA7)|(1<<PA4);
                PORTA = (1<<PA3)|(1<<PA4);
                PORTC = buffer[(out+11*td_t)&0x07FF];
                PORTA = (1<<PA4);
                PORTA = (1<<PA3)|(1<<PA4);
        }
        //Increment the output pointer for the next time we go through the ISR
        out++;
        //Wait for the SPI communication to complete
        while(!(SPSR & (1<<SPIF)));

        //Clear the top two bits of temp, as they are zero
        temp = (temp<<2);
        //Take first two bits of SPDR, as they are the LSBs of the ADC reading
        temp |= (SPDR >> 6);
        //Set CS back high to be ready for the next communication
        PORTB |= (1<<PB4);
        //Store the new data into the buffer
        buffer[new_data&0x07FF] = temp;
        //Increment the new data pointer to be ready for the next ISR
        new_data++;
        //Set PORTD back low for CPU usage measurement
        PORTD &= ~(1<<PD0);
}
```